Daisy Final Project

Credit Card Approval Prediction

Reason why we selected our project

Commercial banks receive a lot of applications for credit cards. Many of them get rejected for many reasons, like high loan balances, low-income levels, or too many inquiries on an individual's credit report, for example. Manually analyzing these applications is mundane, error-prone, and time-consuming. Fortunately, this task can be automated with the power of machine learning, and pretty much every commercial bank does so nowadays. In Daisy_Final_Project,our group of four will build an automatic credit card approval predictor using machine learning algorithm to predict which people are successful in applying for a credit card.

Questions we hope to answer with the data

1. What are the most important parameters for credit card approval?

2. What are the most important parameters for credit card rejection?

3. How many applicants were credit card approval?

4. How many applicants were credit card approval?

MACHINE LEARNING STEPS:

1. Collecting Data
2. Preparing the Data
3. Choosing a Model
4. Training the Model
5. Evaluating the Model
6. Improving the Model
7. Making Predictions

1. Collecting Data

The dataset used in this project is the Credit Card Approval dataset from the [Kaggle]

Credit Card Approval DF

Out[2]:

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	
0	1	30.83	0.000	1	1	Industrials	White	1.25	1	1	1	0	
1	0	58.67	4.460	1	1	Materials	Black	3.04	1	1	6	0	
2	0	24.50	0.500	1	1	Materials	Black	1.50	1	0	0	0	
3	1	27.83	1.540	1	1	Industrials	White	3.75	1	1	5	1	
4	1	20.17	5.625	1	1	Industrials	White	1.71	1	0	0	0	ByOt
...	...	...	...	...	...	...	...	...	...	...	...	...	
685	1	21.08	10.085	0	0	Education	Black	1.25	0	0	0	0	
686	0	22.67	0.750	1	1	Energy	White	2.00	0	1	2	1	
687	0	25.25	13.500	0	0	Healthcare	Latino	2.00	0	1	1	1	
688	1	17.92	0.205	1	1	ConsumerStaples	White	0.04	0	0	0	0	
689	1	35.00	3.375	1	1	Energy	Black	8.29	0	0	0	1	

690 rows × 16 columns

Data Types:

```
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          690 non-null    int64
 1   Age             690 non-null    float64
 2   Debt            690 non-null    float64
 3   Married         690 non-null    int64
 4   BankCustomer    690 non-null    int64
 5   Industry        690 non-null    object
 6   Ethnicity       690 non-null    object
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    int64
 9   Employed        690 non-null    int64
 10  CreditScore     690 non-null    int64
 11  DriversLicense  690 non-null    int64
 12  Citizen         690 non-null    object
 13  ZipCode         690 non-null    int64
 14  Income          690 non-null    int64
 15  Approved        690 non-null    int64
dtypes: float64(3), int64(10), object(3)
memory usage: 86.4+ KB
```

2. Preparing the Data

Data preparation or exploration is the initial step in data analysis, where users explore a large data set in an unstructured way to uncover initial patterns, characteristics, and points of interest. This process isn't meant to reveal every bit of information a dataset holds, but rather to help create a broad picture of important trends and major points to study in greater detail.

1. Putting together all the data and randomizing it. This helps make sure that data is evenly distributed, and the ordering does not affect the learning process.
2. Cleaning the data to remove unwanted data, missing values, rows, and columns, duplicate values, data type conversion, etc.
3. Visualize the data to understand how it is structured and understand the relationship between various variables and classes present.
4. Splitting the cleaned data into two sets - a training set and a testing set. The training set is the set that model learns from. A testing set is used to check the accuracy of model after training.

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
```

In [ ]:
```
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from imblearn.metrics import classification_report_imbalanced
```

In [ ]:
```
# connect to database
```

In [ ]:
```
# Convert to DF
```

In [ ]:
```
# Create our features
#X = df.drop("loan_status", axis=1)
#X = pd.get_dummies(X)
```

In [ ]:
```
# Create our target (target = approved column)
#y = df.loc[:, target].copy()
```

In [ ]:
```
# X.describe() to test
#X.describe()
```

In [ ]:
```
# Check the balance of our target values
#y['loan_status'].value_counts()
```

In [ ]:
```
#from sklearn.model_selection import train_test_split
#X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
#print(Counter(y_train['loan_status']))
#print(Counter(y_test['loan_status']))
```

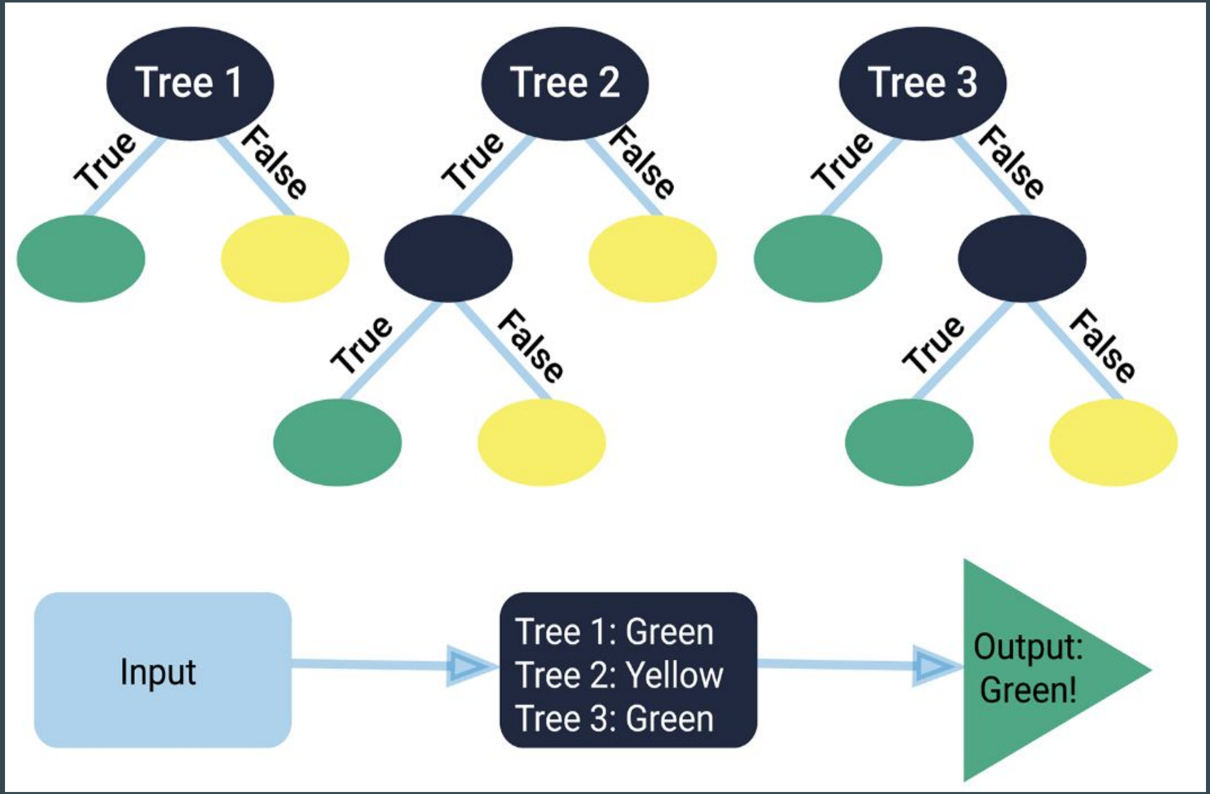
3. Choosing a Model

A machine learning model determines the output we will get after running a machine learning algorithm on the collected data. It is important to choose a model which is relevant to the task at hand.

Random Forest classification algorithm was chosen for Credit Card Approval Prediction Project.

RANDOM FOREST

Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. Random forest algorithm samples the data and build several smaller, simpler decision trees. Each tree is simpler because it is built from a random subset of features.



Random Forest Classifier

In [ ]:
```
#from imblearn.ensemble import BalancedRandomForestClassifier
#rf_model = BalancedRandomForestClassifier(n_estimators=100, random_state=1)
#rf_model.fit(X_train, y_train)
#print(Counter(y_train['loan_status']))
```

In [ ]:
```
# Calculated the balanced accuracy score
#from sklearn.metrics import confusion_matrix, accuracy_score
#y_pred = rf_model.predict(X_test)
#balanced_accuracy_score(y_test,y_pred)
```

In [ ]:
```
# Display the confusion matrix
#cm = confusion_matrix(y_test, y_pred)

# Create a DataFrame from the confusion matrix.
#cm_df = pd.DataFrame(
    #cm, index=["Actual High_Risk", "Actual Low_Risk"], columns=["Predicted High_Risk", "Predicted Low_Risk"])

#cm_df
```

In [ ]:
```
# Print the imbalanced classification report
#print(classification_report_imbalanced(y_test, y_pred))
```

In [ ]:
```
# List the features sorted in descending order by feature importance
#importances = sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)
#for importance in importances:
    #print(f'{importance[1]}:  {importance[0]*100:.1f}%')
```

4. Training the Model

Training is the most important step in machine learning. In training, we will pass the prepared data to our machine learning model to find patterns and make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting.

In [24]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
print(Counter(y_train['Approved']))
print(Counter(y_test['Approved']))
```

```
Counter({0: 285, 1: 232})
Counter({0: 98, 1: 75})
```

Random Forest Classifier

In [25]:

```
from imblearn.ensemble import BalancedRandomForestClassifier
rf_model = BalancedRandomForestClassifier(n_estimators=100, random_state=1)
rf_model.fit(X_train, y_train)
print(Counter(y_train['Approved']))
```

```
Counter({0: 285, 1: 232})
```

5. Evaluating the Model

After training our model, we have to check to see how it's performing. This is done by testing the performance of the model on previously unseen data. The unseen data used is the testing set that we split our data into earlier. If testing was done on the same data which is used for training, we will not get an accurate measure, as the model is already used to the data, and finds the same patterns in it, as it previously did. This will give us disproportionately high accuracy.

When used on testing data, we get an accurate measure of how our model will perform and its speed.

Balance Accuracy Score & Confusion Matrix

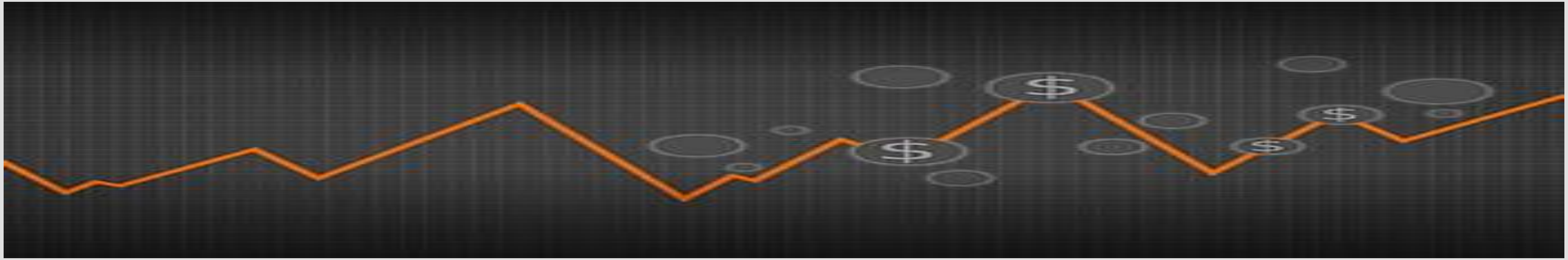
```
0.8819047619047619
```

	Predicted Approved	Predicted Denied
Actual Approved	84	14
Actual Denied	7	68

Imbalance
Classification Report

	pre	rec	spe	f1	geo	iba	sup
0	0.92	0.86	0.91	0.89	0.88	0.77	98
1	0.83	0.91	0.86	0.87	0.88	0.78	75
avg / total	0.88	0.88	0.89	0.88	0.88	0.78	173

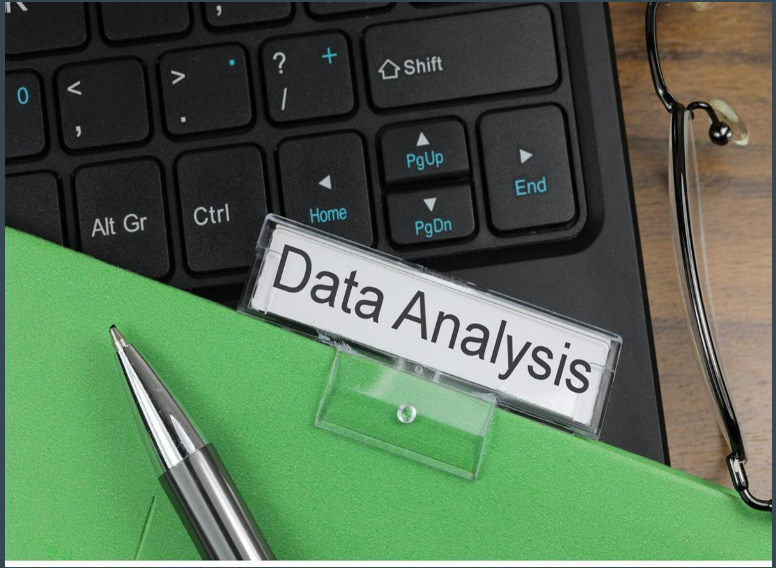
6. Improving the Model



Once we have created and evaluated our model, see if its accuracy can be improved in any way. This is done by tuning the parameters present in our model. Parameters are the variables in the model that the programmer generally decides. At a particular value of our parameter, the accuracy will be the maximum. Parameter tuning refers to finding these values.

7. Making Predictions

In the end, we can use our model on unseen data to make predictions accurately.



In [29]:

```python
# List the features sorted in descending order by feature importance
importances = sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)
for importance in importances:
    print(f'{importance[1]}:  {importance[0]*100:.1f}%')
```

```
PriorDefault:  26.9%
CreditScore:  12.3%
YearsEmployed:  12.2%
Debt:  9.9%
Age:  9.2%
Employed:  5.1%
DriversLicense:  2.0%
Industry_Energy:  1.4%
Gender:  1.4%
Ethnicity_Black:  1.3%
BankCustomer:  1.3%
Married:  1.3%
Industry_Materials:  1.2%
Industry_Utilities:  1.2%
Ethnicity_White:  1.2%
Ethnicity_Asian:  1.0%
Industry_Industrials:  1.0%
Industry_Financials:  1.0%
Citizen_ByBirth:  0.9%
Citizen_Temporary:  0.9%
Industry_ConsumerDiscretionary:  0.9%
Industry_InformationTechnology:  0.9%
Industry_Healthcare:  0.8%
Citizen_ByOtherMeans:  0.8%
Industry_Real Estate:  0.7%
Ethnicity_Latino:  0.7%
Industry_ConsumerStaples:  0.7%
Ethnicity_Other:  0.6%
Industry_CommunicationServices:  0.6%
Industry_Education:  0.4%
Industry_Research:  0.2%
Industry_Transport:  0.1%
```

Daisy Team

Matthew DeLeonardis

Noah Kwiat

Daniel Kim

Valeriia Efremova